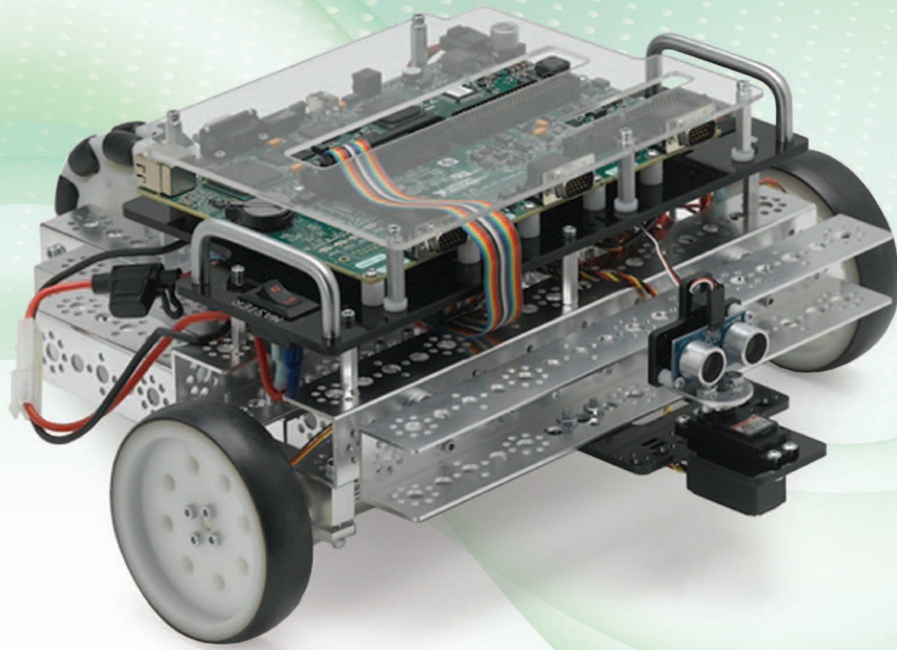

BACKGROUND IMAGE LICENSED BY GRAPHIC STOCK

Teaching Introductory Robotics Programming

Learning to Program with National Instruments' LabVIEW

By Timothy Bower

This article considers strategies for teaching beginning students how to program mobile robots for autonomous operation. Many high school and beginning undergraduate students desire to learn about robotics, but they may lack the required knowledge. Experiences from an undergraduate course are described to illustrate the robot, its programming environment, software design, and algorithms, which faculty can use to guide beginning students from a place of no prior experience to writing impressive, autonomous mobile-robot programs. Autonomous algorithms that perform well and are appropriate for beginning students, including a new wall-following algorithm, are reviewed.

Robotics has become quite popular in education. As the potential for applying robotics to meet real needs expands with the maturation of the technology, so too has interest in learning about robotics. Students of all ages and educational levels want to learn how to build and program robots. The cross-discipline nature of robotics makes it ideal for youths to explore career possibilities in science, technology, engineering, and mathematics (STEM). Affordable hardware options abound for the beginning student wishing to build a simple robot. Sufficient documentation for learning to write a program for the manual operation of robots is also available. Thus, a lack of prior

Digital Object Identifier 10.1109/MRA.2016.2533002
Date of publication: 12 May 2016

1070-9932/16©2016IEEE

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

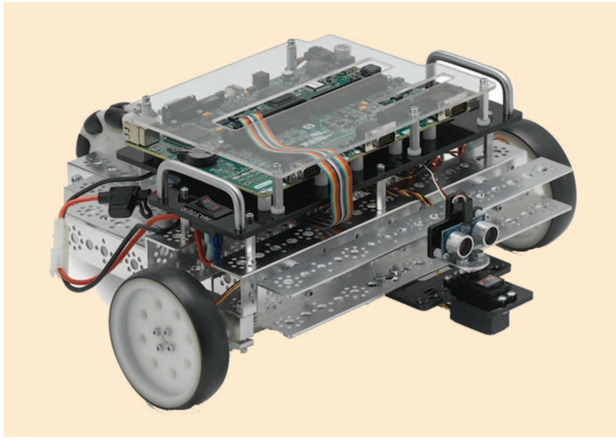

Figure 1. The LabVIEW Robotics Starter Kit, also called the *DaNI Robot*. (Photo courtesy of National Instruments.)

knowledge does not need to prevent students at any age or educational level from pursuing their interest in robotics.

The beginning student's lack of preparation, however, could become a problem when it comes to writing programs to implement autonomous algorithms. Beginning students may not have a sufficient level of programming experience or may not be prepared to understand some of the mathematics of autonomous algorithms. Thus, the robotics programming instructor has a challenge. To be successful, the instructor must challenge beginning students to learn new concepts that are appropriate to their educational level and to exercise their problem solving skills. Zajac [1] points out that mentors and teachers of young and underprepared students must continually hover at the boundary between two states: 1) providing too much assistance in solving problems and 2) providing too little assistance. Students should learn to appreciate the complexities of autonomous algorithms, yet they should also find personal success from implementing simpler algorithms. This article aims to highlight some of the programming and algorithmic solutions appropriate for introducing beginning students to autonomous robot programming.

LabVIEW's simple programming model and graphical capabilities make it a well-suited environment for rapid prototyping and data visualization.

A new course in robotics programming was recently developed at the polytechnic campus of Kansas State University. To allow lower-level undergraduate students to take the course, the prerequisites were limited to a programming course and trigonometry. Based on experiences from the first two offerings of the course, matters related to the robot, the programming environment, software design, and algorithms for introducing beginning students to autonomous mobile-robot programming are considered in this article. Students began the course by writing a sequence of fairly simple programs to operate and test the robot's motors and sensors. As the students learned about robotics, they also learned how to program with LabVIEW. LabVIEW is a trademark of National Instruments (NI) [2], [5]. Later, network programming was used to couple the robot program with a program developed to run on a host computer. Autonomous algorithms could then be studied and implemented. By the end of the semester, each student wrote a program to move the robot to a goal location while avoiding convex and concave obstacles and also completed a self-determined final project. Avoidance of a large concave obstacle, which was referred to as the *cul-de-sac problem*, proved to be an interesting and challenging assignment for the students.

The Development Environment

Because the focus here is on programming robots, it is important that students are able to write robot programs without delay. For this to happen, a prebuilt, dependable robot is needed as well as a programming environment where it is easy to develop, download, run, and debug robot programs. The NI LabVIEW Robotics Starter Kit, also called the *DaNI Robot*, was used [2] (see Figure 1). Using this environment, students were writing robot code by the second week of the semester.

The DaNI Robot Platform

The DaNI Robot uses an NI single-board reconfigurable input/output-embedded controller which has a real-time processor, a user-reconfigurable field-programmable gate array (FPGA), and input/output (I/O) on a single circuit board. The robot's built-in motors and sensors are controlled through the FPGA. Additional I/O ports for analog and digital I/O and pulse width modulation are available if needed. An ethernet port is used for communication between the robot and a host control computer. For wireless communication, a Wi-Fi bridge with an external battery pack is attached to the robot. The robot uses two 4-in wheels in a directional drive configuration with a trailing omni-wheel. Wheel rotation is measured with optical quadrature encoders. The distance to objects is measured with an ultrasonic sensor that is mounted to a servo motor so that the sensor can rotate from side to side.

The LabVIEW Environment

LabVIEW is a graphical programming environment. It uses a graphical model for expressing program logic on block diagrams. Graphical capabilities are also employed for building user interfaces. Each function in a program, which is called a *virtual instrument* (*VI*), consists of a block diagram and a front panel. Some VIs only need a simple front panel to connect the VI's input and output terminals to the block diagram. Other VIs may use a front panel with a graphical user interface and a variety of meters and graphs. LabVIEW's simple programming model and graphical capabilities make it a well-suited environment for rapid prototyping and data visualization.

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

A less obvious but important feature of LabVIEW is that it uses a data-flow model for sequencing the execution of code. This means that a node on a block diagram will execute when data for all of the node's input terminals are available. Nodes are often arranged in a sequential data path where an output from a node is an input to another node; thus, the second node cannot execute until the first node is finished and the data are passed along. However, the sequence of execution for parallel nodes is indeterminate. This behavior occasionally requires special consideration by the programmer; however, capabilities are provided to control the execution order when needed. The advantage of the data-flow model is that it allows LabVIEW to be inherently parallel. When code elements and loops are not in a sequential data path, they can be considered to run as parallel threads of execution.

LabVIEW provides convenient capabilities to synchronize access to critical code sections and to exchange data between parallel threads. The following are two ways synchronization facilities can be used in robotics programming.

- Both the robot and the host controller programs use parallel loops. Various loops generate messages to be sent over the network. The built-in queue facility provides a convenient mechanism for combining the messages into one data stream. Another loop takes messages from the queue and sends them over the network.
- The action engine provides a simple mechanism to synchronize access to critical code sections and to exchange data between threads. An action engine is a VI that can store data using either feedback nodes or uninitialized shift registers. A case structure is used with an enumerator input to the VI for selecting actions to perform when the VI is invoked. Because VIs are, by default, nonreentrant (mutual exclusion), instances of the same action engine VI may be placed in parallel threads to safely share data or access a critical section [3].

As messages from the robot containing sensor data are received, the host controller can use action engines to make calculations and save the data. The autonomous algorithms safely read the data from the action engines as needed. The simplicity of developing programs with parallel threads of execution makes LabVIEW ideal for robotics and especially for teaching robotics programming to beginning students [3], [4]. In the robotics programming class, LabVIEW provided a gentle introduction to parallel programming concepts that would benefit the students in more advanced programming classes. LabVIEW can be viewed as a programming language, but it is really an integrated development environment [5]. LabVIEW makes it easy to program, deploy, and debug robot programs.

LabVIEW Robot Programs

There are two distinct modes for operating robots with LabVIEW. The first mode is useful for debugging programs, is simple to use, and provides students with a convenient mechanism to learn about robot motors and sensors. However, it does not have the real-time performance and program modularity of the second mode. The second mode requires more effort, but results in a better framework for running autonomous programs.

LabVIEW Interactive Mode

When a program intended for a robot platform is initiated on a computer, the robot code is downloaded to the robot and runs. In this mode, the robot is controlled from the computer, and any robot data may be viewed from the computer. This mode has a lot of value for educational and debugging purposes. However, LabVIEW on the computer and on the robot are coordinating extensively, using resources of the robot central processing unit and the network.

Networked Standalone Mode

A LabVIEW program may be compiled and downloaded to the robot to run as a start-up program. Such a program could run as an autonomous application with no interaction with a host controller. However, a purely standalone program has limited functionality. Networking code should be added to the application so that the robot communicates with a host controller program that is running on a computer. The host controller allows a user to control the robot and can also provide another processing resource to augment the robot processor.

In the robotics programming course, the interactive mode was initially used for a sequence of programming and experimentation assignments. In addition to learning about robotics hardware, students developed programs that implemented major algorithmic components for the programs to be developed later in the semester. The reuse of code made the more complex assignments seem less daunting and more like incremental assignments. In the early assignments, students developed the code to drive the robot and to operate the servo motor and ultrasound sensor to collect data for avoiding obstacles. After an introduction to network programming, a framework for future assignments was developed based on the networked standalone mode. To allow immediate sending of data, two TCP connections are made, each used for sending data in one direction. The robot acts as the server waiting for connections from a host controller.

In the first course offering, the host controller only gave operating instructions to the robot. However, in the second course offering, the controller also performed the algorithmic computations. Thus, the robot ran a simple program to control its hardware while taking driving directions from the controller and returning sensor data. Shifting algorithmic processing from the robot to the computer offered performance benefits and made program development easier due to the computer's faster processing capability. All students used the same message-passing application programming interface between the robot and host controller, so it was not necessary to make

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

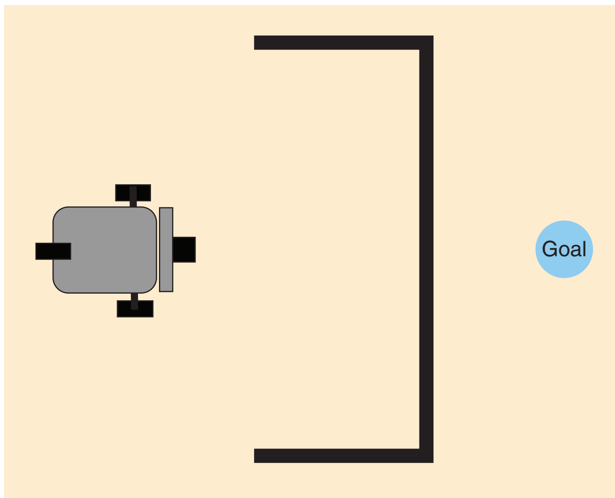

Figure 2. The cul-de-sac problem uses a large concave obstacle. Once inside the cul-de-sac, the robot must temporarily move away from the goal to get out of the cul-de-sac.

any changes to the robot as each student tested his or her program.

> With the exception of a few introductory assignments, each programming assignment contributed code that was used in the solution to the cul-de-sac problem.

After the programming efforts switched from developing programs to run on the robot to programming the host controller, the lectures and assignments shifted to autonomous algorithms. Students developed a sequence of autonomous behaviors for the go-to goal without obstacle avoidance, the go-to goal with avoidance of convex obstacles, wall following, and the cul-de-sac problem. With the exception of a few introductory assignments, each programming assignment contributed code that was used in the solution to the cul-de-sac problem, shown in Figure 2.

Autonomous Algorithms

Autonomous behaviors often have multiple solutions. So, in an introductory course, the instructor might discuss more than one algorithm capable of producing a desired behavior, but direct students to implement algorithms that yield acceptable behavior with concepts appropriate to the students' educational level. Except for wall following, all of the algorithms used in the robotics programming course may be found in literature. In the case of wall following, a new algorithm was developed that was simple for students to understand and implement, yet performed well.

A Comment on PID Controllers

The proportional–integral–derivative (PID) feedback-control system is certainly one of the most important control algorithms used in robotics [6]. PID controllers were discussed in a lecture of the introductory course on robotics programming. However, because the focus of the course is on programming rather than control theory, it was felt that tuning the PID controllers would introduce challenges that are not appropriate for the objectives of the course.

Odometry

Early in the semester, students learned to read odometry data from the optical encoders and convert the data to linear distance traveled within a time interval. Initially, they used the odometry data to track the robot position on the robot platform. Later in the semester, the data were sent to the host controller for processing. The robot's location and orientation in the global coordinate frame is called the *pose* of the robot. The robot's pose consists of its x and y location and its angle of orientation, θ

$$\mathcal{P} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}.$$

Using the distances traveled by the left and right wheels within an interval of time, the change to the robot's pose is calculated using established equations for the kinematics of differential drive robots. Olson provides a nice online primer that helps students understand the kinematics because it works though the calculations in detail [7].

$$d_{\text{moved}} = \frac{d_{\text{left}} + d_{\text{right}}}{2}, \tag{1}$$

$$\psi = \frac{d_{\text{right}} - d_{\text{left}}}{L}, \tag{2}$$

where L is the radius of rotation, which is the distance between the wheels

$$\mathcal{P}_{i+1} = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} + \begin{bmatrix} d_{\text{moved}} \cos\theta_i \\ d_{\text{moved}} \sin\theta_i \\ \psi \end{bmatrix}. \tag{3}$$

Steering the Robot

The right- and left-wheel velocities to steer the robot are determined by the desired forward and rotational velocities, $v(t)$ and $\omega(t)$. L is the distance between the wheels in meters. The velocities are in units of m/s. The rotational velocity, ω, is expressed in rad/s

$$v_r = v + \frac{\omega L}{2}, \tag{4}$$

$$v_l = v - \frac{\omega L}{2}. \tag{5}$$

However, the rotational velocity is not usually a given for steering the robot. Steering is specified by a desired heading, $\phi(t)$ in the global coordinate frame or $\alpha(t) = \phi - \theta$

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

in the robot's local coordinate frame, and a forward velocity, V. This is commonly called the *unicyle model*, which is expressed as a velocity vector in the global coordinate frame, $\mathcal{U}$. The steering controller needs to accept $\mathcal{U}$ as its input and provide (v, ω) as output.

PID Tracking

When the robot's pose is calculated, the point, X, halfway between the drive wheels is used, which moves in-line with the wheels, but not perpendicular to the wheels. Thus it is not possible to directly determine (v, ω) using the velocity vector for X

$$\mathcal{U} = \begin{bmatrix} V\cos\phi \\ V\sin\phi \end{bmatrix} \neq \dot{X} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \end{bmatrix}.$$

So, the common solution is to regulate ω with a PID controller: $\omega = \mathrm{PID}(\alpha = \phi - \theta)$. The wheel velocities can then be calculated from (4) and (5).

Direct Calculation

It is possible for a steering controller to directly calculate reasonable values for (v, ω) from $\mathcal{U}$. Instead of basing the kinematic analysis on the point X, a new point is used that is positioned a small distance, l, directly in front of X, $\tilde{X} = (x + l\cos\theta, y + l\sin\theta)$. This new point, $\tilde{X}$, can move in line with and perpendicular to the wheels. So a controller can be designed to satisfy $\mathcal{U} = \dot{\tilde{X}}$.

$$\begin{aligned} \dot{\tilde{x}} &= \dot{x} - l\dot{\theta}\sin\theta \\ &= v\cos\theta - l\omega\sin\theta \\ \dot{\tilde{y}} &= \dot{y} + l\dot{\theta}\cos\theta \\ &= v\sin\theta + l\omega\cos\theta. \end{aligned}$$

By equating $\mathcal{U}$ to $\dot{\tilde{X}}$, v and ω may be determined

$$\begin{bmatrix} V\cos\phi \\ V\sin\phi \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & l \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$
$$v = V\cos(\phi - \theta)$$
$$\omega = \frac{V}{l}\sin(\phi - \theta).$$

Applying (4) and (5) yields the wheel velocities. Recall that $\alpha = \phi - \theta$, which is the steering heading in the robot's coordinate frame

$$v_r = V(\cos\alpha + K\sin\alpha), \tag{6}$$

$$v_l = V(\cos\alpha - K\sin\alpha). \tag{7}$$

The variable $K = (L/2l)$ can be regarded as a single tunable parameter. With $K = 0.66$, our robots demonstrated responsive, yet smooth and stable steering. The kinematics of an alternate point $\tilde{X}$ are discussed in [8]–[10] and applied to directional drive robots in [11] and [12]. The online study guide used with the robotics programming class shows more detailed analysis of the controller [13]. The students were able to understand most of the math involved in the derivation of the controller equations. They certainly appreciated using a steering controller with a simple, direct calculation and only one tunable parameter.

Obstacle Avoidance

The obstacle-avoidance algorithm used a simple sum of vectors as described in [14]. Using only one ultrasound sensor required refinements to the algorithm, which the students designed. The servo motor rotates side to side stopping at fixed positions where ultrasound measurements are taken. The ultrasound data represent the distance in meters to any object detected. When no object is detected, the maximum range of the sensor (3 m) is used.

> **The students appreciated using a steering controller with a simple, direct calculation and only one tunable parameter.**

Ultrasound measurements are taken at angles of {–1.5, –1, –0.5, 0, 0.5, 1, 1.5} rad. Using the measurements as vectors, the sum of the vectors is calculated to yield the "avoid-obstacle" heading (see Figure 3). Due to the symmetry of the measurement angles, $\alpha_{ao} = 0$, when no obstacles are detected, shorter vectors from obstacle detection cause the sum of vectors to deflect away from the obstacle. The simple sum of vectors from the ultrasound data was a good start, but the resulting heading was not adequate. This provided an

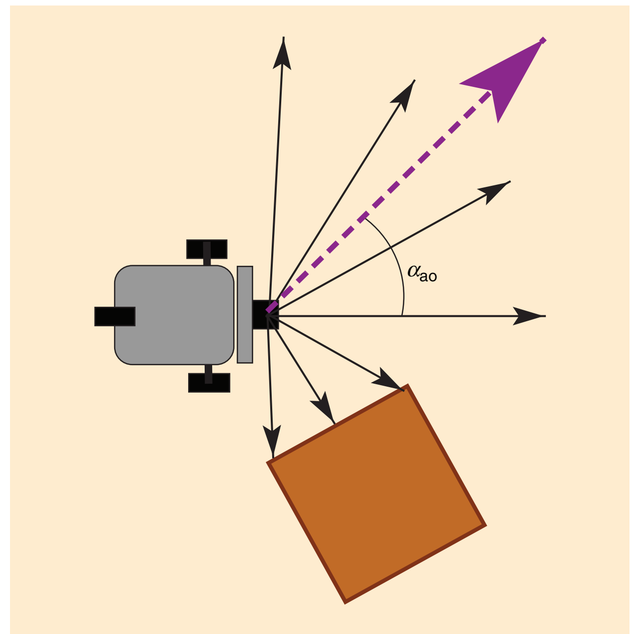

Figure 3. The sum of ultrasound measurement vectors yields the "avoid-obstacle" heading, α_{ao}.

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

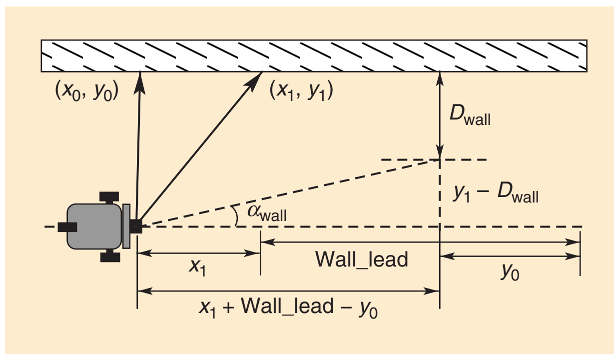

Figure 4. An abstract right triangle for computing the wall-following heading.

opportunity for the students to observe, analyze, and collaborate to find solutions:

- It detects objects to the side that pose no collision threat.
  - The solution was to limit the lengths of the ultrasound data linearly by measurement angle so that forward measurements have greater range than side measurements.
- The sum of the vectors always points forward and does not adequately deflect away from obstacles.
  - The solution was to shift each vector in the opposite direction. Thus, detected obstacles can result in negative vectors. A panic element to the shift was also adopted to give larger negative vectors in the event of a near collision.

After students successfully demonstrated their program to solve the cul-de-sac problem, they consistently expressed a sense of accomplishment.

Autonomous Behaviors

One of the parallel executing loops of the host controller program contains a finite state machine (FSM) where the robot's algorithmic behaviors are implemented. With each loop iteration, an updated heading and velocity is calculated so that the robot drives according to a specified behavior. The FSM is a natural construct for implementing autonomous behaviors. Implementing the FSM was also a valuable educational experience for the students.

Blending Headings

In both the go-to goal and wall-following behaviors with obstacle avoidance, the final steering heading is a blended combination of two angles: α_{ao} for avoid-obstacle and β for the behavior component. When there is little threat of a collision, $|\alpha_{ao}| \approx 0$ and the behavior component dominate. When the risk of a collision is increased, $|\alpha_{ao}|$ becomes larger and the robot is diverted. We define a weighting variable, h:

$$h = \begin{cases} \dfrac{|\alpha_{ao}|}{\text{Threshold}} & \text{if } |\alpha_{ao}| < \text{Threshold} \\ 1 & \text{otherwise} \end{cases}, \tag{8}$$

$$\alpha = h \cdot \alpha_{ao} + (1 - h) \cdot \beta. \tag{9}$$

Pure obstacle avoidance, $\alpha = \alpha_{ao}$, occurs when $|\alpha_{ao}| \geq$ Threshold. The Threshold is a tunable parameter for each behavior.

Go-To Goal

The go-to goal behavior uses the robot's pose and the location of a goal. A simple trigonometry calculation provides the direction to the goal, ϕ_g. The robot heading is then simply $\alpha_g = \phi_g - \theta$. Adding blended obstacle avoidance simply requires using the VI to compute the steering heading with (8) and (9) using $\beta = \alpha_g$ and Threshold $= \pi/4$. The only difficulty that any students had with this quick assignment was making sure that the robot stopped at the goal.

Wall Following

Autonomous controlled mobile robots require a wall-following behavior to maneuver around large concave obstacles. Convex obstacles may be avoided using a blended go-to goal with avoid-obstacle behavior. However, when inside a large concave obstacle, such as a cul-de-sac, the go-to goal-based behavior causes the robot to become stuck in a local minimum, where it wanders in the cul-de-sac but refuses to drive away from the goal as is needed to exit the cul-de-sac. When a wall-following behavior is used, the robot can follow the contours of the obstacle to move around it and then proceed toward the goal location. The algorithm used to achieve wall-following behavior uses the blended avoid-obstacle concept along with a behavioral heading, α_{wall}, which follows the contours of the wall. The algorithm for computing the behavioral heading was original. It was developed specifically to be a simple algorithm for students to understand and implement. Despite its simplicity, the overall algorithm has demonstrated good performance for the intended application.

The behavioral heading calculation uses an abstract right triangle, shown in Figure 4. The side lengths of the triangle derive from the first two ultrasound measurements from the side of the robot facing the wall. The wall-following heading is then simply an angle calculation from the triangle. Using the known measurement angles, the first two measurements are converted to points in the robot's local coordinate frame. The wall-following heading is given by

$$\rho = a\tan 2((|y_1| - D_{\text{wall}}), (x_1 + \text{Wall_lead} - |y_0|))$$

$$\alpha_{\text{wall}} = \begin{cases} \rho & \text{if the wall is to the left} \\ -\rho & \text{if the wall is to the right.} \end{cases}$$

The relationship of $|y_1|$ to D_{wall} reports the robot's current position relative to what is detected about the approaching

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.

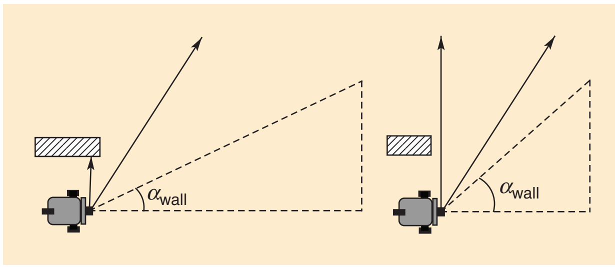

Figure 5. At the end of a wall, the term y_0 modulates the side length of the abstract triangle and thus the turning heading.

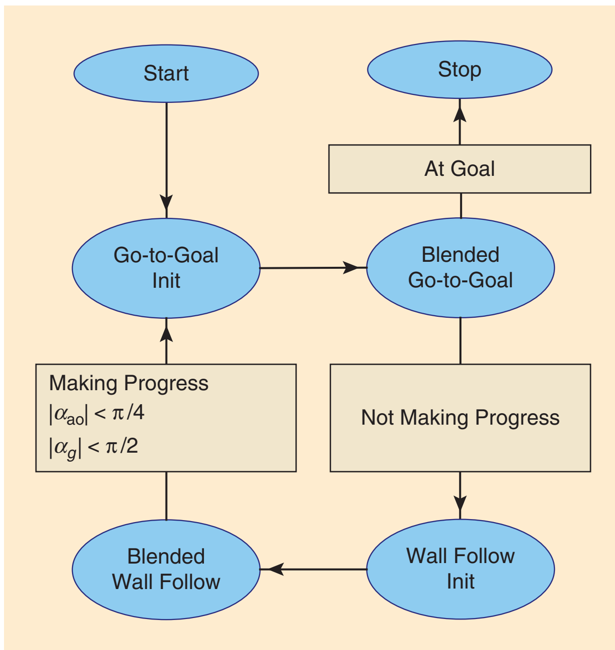

Figure 6. State transitions for go-to goal with avoidance of convex and concave obstacles.

contour of the wall. The x_1 and Wall_lead terms serve as damping factors in the equation. The $|y_0|$ term strongly influences the heading when the wall ends or turns away from the robot; see Figure 5.

The wall-following heading and the avoid-obstacle heading complement each other. The wall-following component is most adept at keeping the robot from getting too far from the wall and induces the turning when the wall turns away from the robot. Conversely, the avoid-obstacle heading diverts the robot when it or its projected path is too close to the wall. The steady-state distance from the wall is where the wall-following and avoid-obstacle headings offset. On our system, D_{wall} was set at 0.4 m, and the robot tracked walls at a range of approximately 0.5 m. Adding the blended obstacle avoidance simply requires the use of VI to compute the steering heading, with (8) and (9) using $\beta = \alpha_{wall}$ and Threshold $= \pi/2$. On our system, the value of 2 m worked well for Wall_lead.

The Cul-De-Sac FSM

After completing the algorithmic behaviors, students only needed to add state transitions to the FSM to solve the cul-de-sac problem, as shown in Figure 6. Both the blended go-to goal and wall-following functions have initialization states that save the distance to the goal in a make-progress action engine. Making progress is defined as the current distance to the goal being within 10 cm of the previous nearest position to the goal. The wall-following initialization state also determines if the goal, and thus also the wall, is to the left or right of the robot. After students successfully demonstrated their program to solve the cul-de-sac problem, they consistently expressed a sense of accomplishment for writing a fairly challenging autonomous mobile robot program. A short video of a robot going to a goal while avoiding a cul-de-sac obstacle may be viewed online at https://www.youtube.com/watch?v=OyqRtPnqP7w.

References

[1] R. Zajac, "Mentoring middle school NXT robotics: Math development as a primary design constraint," presented at *Proc. ASEE Midwest Section Conf.*, Salina, KS, 2013.

[2] National Instruments. (2015). *NI LabVIEW Robotics Starter Kit Website.* [Online]. Available: http://www.ni.com/datasheet/pdf/en/ds-217

[3] T. Bress, *Effective LabVIEW programming*. Allendale, NJ: NTS Press, 2013.

[4] R. Bitter, *LabVIEW advanced programming techniques*. Boca Raton, FL: CRC Press/Taylor & Francis, 2007.

[5] National Instruments. (2013). *What is LabVIEW*? [Online]. Available: http://www.ni.com/newsletter/51141/en/

[6] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ: Princeton Univ. Press, 2012.

[7] E. Olson. (2005). *A Primer on Odometry and Motor Control.* [Online]. Available: http://web.mit.edu/6.186/2006/doc/odomtutorial/odomtutorial.pdf

[8] J. Pomet, B. Thuilot, G. Bastin, and G. Campion, "A hybrid strategy for the feedback stabilization of nonholonomic mobile robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, Nice, France, May 1992, pp. 129–134.

[9] K. Amar and S. Mohamed, "Stabilized feedback control of unicycle mobile robots," *Int. J. Advanced Robotic Syst.*, vol. 10, no. 187, Apr. 2013.

[10] J. Lawton, B. Young, and R. Beard, "A decentralized approach to elementary formation maneuvers," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, Apr. 2000, pp. 2728–2733.

[11] P. Ögren, "Formations and obstacle avoidance in mobile robot control," Ph.D. dissertation, Roy. Inst. Technol., Stockholm, Sweden, 2003.

[12] M. Egerstedt. (2014) *Control of Mobile Robots, Lecture 7.4: A Clever Trick* [Online]. Available: https://class.coursera.org/conrob-002/wiki/Week7a

[13] T. Bower. (2015). *Robotics Programming Study Guide: A Clever Trigonometry-Based Controller.* [Online]. Available: http://faculty.salina.k-state.edu/tim/robotics_ sg/Control/controllers/trig_ trick.html

[14] K. Agarwal, S. Mahtab, S. Bandyopadhyay, and S. Das Gupta, "A proportional-integral-derivative control scheme of mobile robotic platforms using matlab," *IOSR J. Elect. Electron. Eng.*, vol. 7, pp. 32–39, Oct. 2013.

Timothy Bower, associate professor of computer systems technology, Polytechnic Campus, Kansas State University, Salina, 67401. E-mail: tim@ksu.edu.

Authorized licensed use limited to: Kansas State University. Downloaded on May 16,2022 at 21:53:52 UTC from IEEE Xplore. Restrictions apply.